php基础课程文档

课程介绍

现在是2023年9月，本系列课程基于php8，但是现在市面上大多数还是php7的，所以我们讲的大部分知识都是7和8通用的，后面会专门讲8相对于7的特性和区别等。
课程包含基础知识和实战项目：
正所谓纸上得来终觉浅，绝知此事须躬行，只学习基础知识很容易忘记，但结合实战项目就能记得很牢固，所以每隔几节课就会有一个小项目来巩固知识。

关于知识星球：
本课程是完全公开且免费的，除了课程外，还有专门交流和写作业的社区，那就是猫叔的星球（https://maoshu.fun ），里面会总结一些学员常见的问题，还会布置作业，可以在里面提交答案，然后我来分析代码有没有问题或者有更优的处理方法。这种作业的形式，可以让你真正学会，而不是眼睛会了手不会。

除此之外，猫叔还提供一对一带徒陪跑服务，这个是收费的，价格很亲民，有需要的话请详细咨询我。咱们的课程是23年9月开始边录边发的，所以有问题或者建议可以在评论区提，我会在后面的课程中做优化。

学习网站推荐：
PHP 教程 (w3school.com.cn)
PHP 教程 | 菜鸟教程 (runoob.com)

关于php

- PHP 是 "PHP Hypertext Preprocessor" 的首字母缩略词
- **PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。**
- **PHP 是开源免费的，并且使用广泛**，可供免费下载和使用，可商用
- **PHP 脚本在服务器上执行**
- **PHP 是解释型语言，按顺序从上往下执行，无需编译，直接运行**
- 它强大到足以成为在网络上最大的博客系统的核心（WordPress）！
- 它深邃到足以运行最大的社交网络（facebook）！
- 而它的易用程度足以成为初学者的首选服务器端语言！

什么是 PHP 文件?

- PHP 文件能够包含文本、HTML、CSS 以及 PHP 代码
- PHP 文件的后缀是 ".php"

- PHP 代码在服务器上执行，而结果以纯文本返回浏览器

PHP 能够做什么?

- PHP 能够生成动态页面内容 （Html页面渲染）
- PHP 能够创建、打开、读取、写入、删除以及关闭服务器上的文件 （文件操作）
- PHP 能够接收表单数据 （例如注册登录）
- PHP 能够发送并取回 cookies （记录用户状态等信息）
- PHP 能够添加、删除、修改数据库中的数据 （数据操作）
- PHP 能够限制用户访问网站中的某些页面 （权限管理）
- PHP 能够对数据进行加密和压缩
- PHP 通常用于小程序 app 浏览器插件等的后端服务接口

通过 PHP，您可以不受限于只输出 HTML。您还能够输出图像、PDF 文件、甚至 Flash 影片。您也可以输出任何文本，比如 XHTML 和 XML。

为什么使用 PHP?

- PHP 是跨平台的，可运行于各种平台（Windows, Linux, Unix, Mac OS X 等等）
- PHP 兼容几乎所有服务器（Apache，Nginx, IIS 等等）
- PHP 支持多种数据库 (Mysql)
- PHP 是免费的。请从官方 PHP 资源下载：www.php.net
- PHP 易于学习，并可高效地运行在服务器端

最好应具备的基础知识

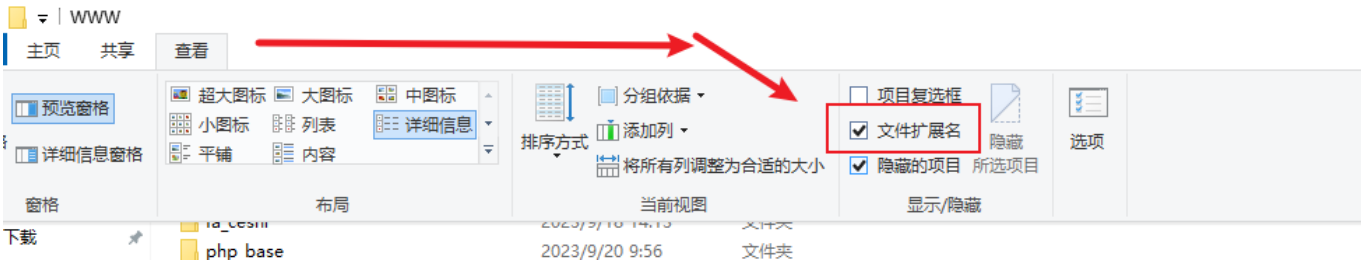
在继续学习之前，您需要对下面的知识有基本的了解：

- HTML
- CSS
- JavaScript（入门级这个无所谓）

环境安装和编辑器推荐

编辑器根据自己喜好即可，我推荐vscode /notepad-- /notepad next /phpstorm
环境我推荐使用phpstudy（小皮）
演示创建一个站点

注意：把电脑文件扩展名显示出来



php语法

基础 PHP 语法

PHP 脚本以 `<?php` 开头，以 `?>` 结尾：

```
<?php // 此处是 PHP 代码 ?>

<?php
// 此处是 PHP 代码
?>
```

PHP 脚本可放置于文档中的**任何**位置，文件扩展名是 ".php"。
PHP 文件通常包含 HTML 标签以及一些 PHP 脚本代码，注意： html js css可以写在php文件里运行，但php不能写在html js css文件里运行。

PHP 语句以分号结尾（;）

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一张 PHP 页面</h1>

<?php
echo "Hello maoshu!";
?>

</body>
</html>
```

输出方法：echo 和 print

不同点：

- echo - 能够输出一个以上的字符串，英文逗号隔开

- print - 只能输出一个字符串，并**始终返回 1**
- echo 比 print 稍快，并且开销低

```php
<?php
echo "<h2>猫叔123!</h2>";
echo "Hello world!<br>";
echo "Maoshu", " string", 1, " 2 ";
?>
```

相同点：

- 都是一个语言结构，有无括号均可使用：echo 或 echo() print 或 print()

注释

注释不会被作为程序来读取和执行。它唯一的作用是供代码编辑者阅读（让别人阅读明白，提醒自己做过什么，特别是一些函数方法的用途等）

```php
<?php
// 这是单行注释

# 这也是单行注释

/*
这是多行注释块
它横跨了
多行
随意换行没问题
echo '猫叔';
*/
?>
```

变量

变量是存储信息的容器，有点类似初中数学里的代数 `x=1,y=2`

```php
$a=5;
$_a = 5;
$b=6;
$c=$a+$b;
echo $c;
```

=号并不是真实的等号，而是叫赋值

变量命名规则

- 变量以 $ 符号开头，其后是变量的名称
- 变量名称必须以字母或下划线开头
- 变量名称不能以数字开头，不能有空格
- 变量名称只能包含字母 数字 字符和 下划线（A-z、0-9 以及 _）
- 变量名称对大小写敏感（$y 与 $Y 是两个不同的变量）

注释：PHP 变量名称对大小写敏感!

推荐的几种命名方法：

- 下划线命名法 `$first_name = 'zhang';`
- 小驼峰命名法 `$firstName = 'zhang';`
- 大驼峰命名法 `$FirstName = 'zhang';`
-

```
$txt="Hello world!";
$a=5;
$b=8.5;
```

不必告知 PHP 变量的数据类型，php会根据它的值，自动把变量转换为正确的数据类型

变量也有**作用域**之分，等到后面函数的时候再说。

空白符

```
echo '猫叔

你在干什么

?

';

/**

我在

看抖音小姐姐
```

```
*/
```

大小写敏感

所有用户定义的函数、类和关键词（例如 if、else、echo 等等）都对大小写**不敏感**，
但是所有变量都对大小写**敏感**

```
ECHO "Hello 猫叔!<br>";
echo "Hello 猫叔!<br>";

EcHo "Hello 猫叔!<br>";
PRint "Hello 猫叔!<br>";
```

```
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
```

命令行的使用

可以像python一样在命令行里运行php
用cmd或者vscode插件

可以获取用户输入的内容

```
$input = readline("请输入内容: ");
echo "您输入的内容是:  " . $input;
```

```
echo "请输入内容: ";
$input = fgets(STDIN);
echo "您输入的内容是:  " . $input;
```

数据类型

字符串、整数、浮点数、逻辑（布尔型）、数组、对象、NULL、资源类型
用var_dump() 会返回变量的数据类型和值，一般用于开发调试时使用

只获取数据类型 `echo gettype($a);`

字符串

字符串是字符序列，比如 "Hello world!"。

字符串可以是引号内的任何文本，可以使用单引号或双引号
注意双引号和单引号的区别

```php
$x = "Hello world!";
echo $x;
echo "<br>";
$x = 'Hello world!';
var_dump($x);
```

整数

和数学里的整数有些区别

- 整数必须有至少一个数字（0-9）
- 整数不能包含逗号或空格
- 整数不能有小数点
- 整数正负均可
- 可以用三种格式规定整数：十进制、十六进制（前缀是 0x）或八进制（前缀是 0）

```php
$x = 5985;
var_dump($x);
echo "<br>";
$x = -345; // 负数
var_dump($x);
echo "<br>";
$x = 0x8C; // 十六进制数
var_dump($x);
echo "<br>";
$x = 047; // 八进制数
var_dump($x);
```

浮点数

浮点数是有小数点或指数形式的数字

```php
$x = 10.365;
var_dump($x);
echo "<br>";
$x = 2.4e3;
var_dump($x);
echo "<br>";
```

```php
$x = 8E-5;
var_dump($x);
```

逻辑（布尔值）

```php
$x=true;
$y=false;
```

数组

在一个变量中存储多个值
分为一维数组和多维数组，认识即可，不要深究，后面会专门讲

```php
$cars=array("Volvo","BMW","SAAB");
$_cars=["Volvo","BMW","SAAB"];
var_dump($cars);
var_dump($_cars);
```

对象

需要用到类和封装的知识
等学到类的时候再讲

Null

特殊的 NULL 值表示变量无值。NULL 是数据类型 NULL 唯一可能的值
注意：可以通过设置变量值为 NULL 来清空变量数据

```php
$x="Hello world!";
var_dump($x);

$x=null;
var_dump($x);
```

resource 资源类型

等学到数据库的时候再讲

初级实战 - 个人博客的开头部分

实战目标：
html简单介绍
复习变量声明和输出
复习注释
了解变量重复赋值

EOF(heredoc) 定界符使用说明

- 以 <<<EOF 开始标记开始，以 EOF 结束标记结束
- 结束标记必须顶头写，独自占一行，不能有缩进和空格
- 在结束标记末尾要有分号
- **EOF** 可以用任意其它字符代替，开始标记和结束标记相同即可，比如常用大写的 EOT、EOD、EOF 来表示，但是不只限于那几个(也可以用：JSON、HTML等)，只要保证开始标记和结束标记不在正文中出现即可。
- 位于开始标记和结束标记之间的变量可以被正常解析，但是函数则不可以。在 heredoc 中，变量不需要用连接符 . 或 , 来拼接
- 当内容需要内嵌引号（单引号或双引号）时，不需要加转义符，本身对单双引号转义。

```
<?php
$name="变量会被解析";
$a=<<<EOF
$name<br>
看上面的name已经不是name啦！！！<br>
EOF;
echo $a;

echo '<hr>';

$a=<<<EOF
<a>html格式会被解析</a><br/>你看看a标签显示了没？
你看看br显示了没？
<br>
EOF;
echo $a;

echo '<hr>';

echo <<<EOF
"双引号外所有被排列好的格式都会被保留"

"但是双引号内会保留转义符的转义效果,比如table:\t和换行：\n下一行"

你看页面里有\t吗？
```

```
\n \t 和引号没关系吧？

那单'引'号呢？'\t \n 是否能看到？'
EOF;
```

运算符

> 注：资料来源于菜鸟教程 www.runoob.com

算术运算符

运算符	名称	描述	实例	结果
x + y	加	x 和 y 的和	2 + 2	4
x - y	减	x 和 y 的差	5 - 2	3
x * y	乘	x 和 y 的积	5 * 2	10
x / y	除	x 和 y 的商	15 / 5	3
x % y	模（除法的余数）	x 除以 y 的余数	5 % 2 10 % 8 10 % 2	1 2 0
-x	设置负数	取 x 的相反符号	<?php $x = 2; echo -$x; ?>	-2
~x	取反	x 取反，按二进制位进行"取反"运算。 运算规则： ~1=-2; ~0=-1;	<?php $x = 2; echo ~$x; ?>	-3
a . b	并置	连接两个字符串	"Hi" . "Ha"	HiHa

```php
<?php
$x=10;
$y=6;
echo ($x + $y); // 输出16
echo '<br>';  // 换行

echo ($x - $y); // 输出4
```

```php
echo '<br>';  // 换行

echo ($x * $y); // 输出60
echo '<br>';  // 换行

echo ($x / $y); // 输出1.6666666666667
echo '<br>';  // 换行

echo ($x % $y); // 输出4
echo '<br>';  // 换行

echo -$x;

//整数之间的整除，参数也必须是整数，向下取整
var_dump(intdiv(10, 3));
?>
```

赋值运算符

运算符	等同于	描述
x = y	x = y	左操作数被设置为右侧表达式的值
x += y	x = x + y	加
x -= y	x = x - y	减
x *= y	x = x * y	乘
x /= y	x = x / y	除
x %= y	x = x % y	模（除法的余数）
a .= b	a = a . b	连接两个字符串

```php
<?php
$x=10;
echo $x; // 输出10

$y=20;
$y += 100;
$y = $y + 100;
echo $y; // 输出120
```

```php
$z=50;
$z -= 25;
echo $z; // 输出25

$i=5;
$i *= 6;
echo $i; // 输出30

$j=10;
$j /= 5;
echo $j; // 输出2

$k=15;
$k %= 4;
echo $k; // 输出3
?>
```

递增/递减运算符

运算符	名称	描述
++ x	预递增	x 加 1，然后返回 x
x ++	后递增	返回 x，然后 x 加 1
-- x	预递减	x 减 1，然后返回 x
x --	后递减	返回 x，然后 x 减 1

```php
<?php
$x=10;
echo ++$x; // 输出11

$y=10;
echo $y++; // 输出10

$z=5;
echo --$z; // 输出4

$i=5;
echo $i--; // 输出5
?>
```

比较运算符

运算符	名称	描述	实例
x == y	等于	如果 x 等于 y，则返回 true	5==8 返回 false
x === y	绝对等于	如果 x 等于 y，且它们类型相同，则返回 true	5==="5" 返回 false
x != y	不等于	如果 x 不等于 y，则返回 true	5!=8 返回 true
x <> y	不等于	如果 x 不等于 y，则返回 true	5<>8 返回 true
x !== y	不绝对等于	如果 x 不等于 y，或它们类型不相同，则返回 true	5!=="5" 返回 true
x > y	大于	如果 x 大于 y，则返回 true	5>8 返回 false
x < y	小于	如果 x 小于 y，则返回 true	5<8 返回 true
x >= y	大于等于	如果 x 大于或者等于 y，则返回 true	5>=8 返回 false
x <= y	小于等于	如果 x 小于或者等于 y，则返回 true	5<=8 返回 true

```php
<?php
$x=100;
$y="100";

var_dump($x == $y);
echo "<br>";
var_dump($x === $y);
echo "<br>";
var_dump($x != $y);
echo "<br>";
var_dump($x !== $y);
echo "<br>";

$a=50;
$b=90;

var_dump($a > $b);
echo "<br>";
var_dump($a < $b);
?>
```

逻辑运算符

运算符	名称	描述	实例
x and y	与	如果 x 和 y 都为 true，则返回 true	x=6 y=3 (x < 10 and y > 1) 返回 true
x or y	或	如果 x 和 y 至少有一个为 true，则返回 true	x=6 y=3 (x==6 or y==5) 返回 true
x xor y	异或	如果 x 和 y 有且仅有一个为 true，则返回 true	x=6 y=3 (x==6 xor y==3) 返回 false
x && y	与	如果 x 和 y 都为 true，则返回 true	x=6 y=3 (x < 10 && y > 1) 返回 true
x || y	或	如果 x 和 y 至少有一个为 true，则返回 true	x=6 y=3 (x==5 || y==5) 返回 false
! x	非	如果 x 不为 true，则返回 true	x=6 y=3 !(x==y) 返回 true

and 和 && 的区别， or 和 || 的区别

主要体现在优先级：

- && 运算符的优先级比 and 运算符高。
- 这意味着，在一个表达式中， && 运算符会先被执行，然后才是 and 运算符。

```php
<?php
// 在表达式中使用&&运算符
$bool = true && false;

// 显示&&运算符的运算结果
echo "&&运算符的结果为: ";
if($bool){
    echo 'true';
}
else{
    echo 'false';
}

echo '----';
```

```php
// 在表达式中使用and运算符
$bool = true and false;
// 显示and运算符的运算结果
echo "and运算符的结果为：";
if($bool){
    echo 'true';
}
else{
    echo 'false';
}
?>
```

阻断效果

```php
//or-前面语句值为真，or后面不执行；否则，执行
$result = 0 or var_dump('执行我的语句');  //输出-执行我的语句
var_dump($result);  //int 0
$result = 2 or var_dump('执行我的语句');  //不输出
var_dump($result);  //int 2
```

条件语句 (条件分支)

if语句

仅当指定条件成立时执行代码

```php
if (条件)
{
    条件成立时要执行的代码;
}
```

if...else...语句

在条件成立时执行一块代码，条件不成立时执行另一块代码

```php
if (条件)
{
    条件成立时执行的代码;
}
else
{
```

```
    条件不成立时执行的代码;
}
```

if...elseif....else

```
if (条件)
{
    if条件成立时执行的代码;
}
elseif (条件)
{
    elseif条件成立时执行的代码;
}
elseif (条件)
{
    elseif条件成立时执行的代码;
}
else
{
    条件不成立时执行的代码;
}
```

switch

注意：break很重要，不要漏掉

```
<?php
switch (n)
{
        case label1:
            如果 n=label1，此处代码将执行;
            break;
        case label2:
            如果 n=label2，此处代码将执行;
            break;
        default:
            如果 n 既不等于 label1 也不等于 label2，此处代码将执行;
}
?>
```

什么时候适合用switch?
适合条件比较多，并且条件单一固定值匹配时用

循环

让相同的代码块一次又一次地重复运行

for 循环

用于预先知道脚本需要运行的次数的情况

```
for (初始值; 条件; 增量)
{
    要执行的代码(叫循环体);
}
```

参数：

- **初始值**：主要是初始化一个变量值，用于设置一个计数器（但可以是任何在循环的开始被执行一次的代码）。
- **条件**：循环执行的限制条件。如果为 TRUE，则循环继续。如果为 FALSE，则循环结束。
- **增量**：主要用于递增计数器（但可以是任何在循环的结束被执行的代码）。

注释：上面的**初始值**和**增量**参数可为空，或者有多个表达式（用逗号分隔）。

```
for ($i=1,$j=0; $i<=5; )
{
    echo "数字为 " . $i.$j . PHP_EOL;
    $i++;
}
```

while 循环

只要指定的条件成立，则循环执行代码块，直到指定的条件不成立

```
while (条件)
{
    要执行的代码;
}
```

注意避免死循环

```
$i=1;
while($i<=5)
{
```

```
    echo "The number is " . $i . "<br>";
    $i++;
}
```

do...while

语句会至少执行一次代码，然后检查条件，只要条件成立，就会重复进行循环

```
do
{
    要执行的代码;
}
while (条件);
```

```
$i=1;
do
{
    $i++;
    echo "The number is " . $i . "<br>";
}
while ($i<=5);
```

continue和break

continue

```
  for ($i=0; $i < 5; $i++) {
    if($i == 3){
        continue;
    }
    echo $i;
  }
```

break

```
for ($i=0; $i < 5; $i++) {
    if($i>3){
        break;
    }
    echo $i;
}
```

break 2

```
for ($i = 0; $i < 3; $i++) {
        echo "外层循环: " . $i . "<br>";

        for ($j = 0; $j < 5; $j++) {
            echo "内层循环: " . $j . "<br>";

            if ($j == 2) {
                break 2; // 跳出两层循环
            }
        }
 }

echo '已跳出全部循环';
```

在while中使用continue break和break2

```
    $a = 10;
    while ($a <= 20) {
        $a++;

        if($a == 15){
            //continue;
            break;
        }

        echo $a.'---';
    }
```

```
    $a = 10;
    while ($a <= 20) {
        $a++;

        if($a == 15){
            for ($i=0; $i < 5; $i++) {
                if($i==3){
                    break 2;
                }
                echo '<br>'.$i.'<br>';
            }
        }
```

```
        echo $a.'---';
    }
```

foreach 循环

用于遍历数组，等学数组一起讲

数组

数组能够在单个变量中存储多个值，并且可以根据 键 访问其中的 值

数值数组

```
$maoshu = array("猫叔1","猫叔2","猫叔3","猫叔4","猫叔5");
$maoshu = ["猫叔1","猫叔2","猫叔3","猫叔4","猫叔5"];

var_dump($maoshu[1]);
var_dump($maoshu[5]);
```

```
$maoshu2 = [];
$maoshu2[3] = '3';

var_dump($maoshu2);
```

关联数组

关联数组是使用您分配给数组的指定的键的数组

```
$age=["maoshu"=>"35","dong"=>"37"];

//或者

$age['maoshu']="35";
$age['dong']="37";
```

获取数组的长度 - count() 函数

```
echo count($maoshu);
```

遍历数组

```
for($i=0;$i<count($maoshu);$i++) {
        echo $maoshu[$i];
        echo "<br>";
}

foreach ($maoshu as $value)
{
    要执行代码;
}

foreach ($maoshu as $key => $value)
{
    要执行代码;
}
```

for可以，while也可以，但最常用的是foreach

多维数组

多维数组是包含一个或多个数组的数组

```
  $maoshuArr = [
    [
        "免费课"=>[
            "猫叔课",
            "php",
            "python"
        ],
        "体验课"=>[
            "php",
            "js",
            "vue"
        ]
    ],
    [
        "收费课"=>[
            "油猴脚本",
            "浏览器插件",
            "666"
        ]
    ]
  ];
```

```
print_r( $maoshuArr );
```

```
$maoshuArr = [
  "test" => [
       "免费课"=>[
           "猫叔课",
           "php",
           "python",
       ],
       "体验课"=>[
           "php",
           "js",
           "vue",
       ]
   ],
  "product" => [
       "免费课"=>[
           "产品1",
           "产品2",
       ],
       "体验课"=>[
           "产品3",
           "产品4",
       ]
   ]
];

print_r( $maoshuArr );
```

遍历多维数组

```
foreach ($maoshu as $key => $value)
{
    要执行代码;
        foreach ($value as $kk => $vv)
        {
            要执行代码;
        }
}
```

初级实战2 - 个人博客的开头部分

实战目标:

- 复习运算符的使用
- 复习条件语句的使用
- 复习循环
- 学会使用数组

任务：
循环数组：输出顶部导航栏，输出博客标题和时间
条件语句：如果为第一个导航，则给出突出显示
运算：输出数字的大小从大到小排列

函数

函数是一段可重复使用的代码块，可以将一系列操作封装起来，使代码更加模块化、可维护和可重用，来大大节省我们的开发时间和代码量，提高编程效率。

内置函数

内置函数有很多，之前学的print_r() count() 就是一个内置函数，
获取当前时间 time()，date('Y-m-d H:i:s')

这节我们主要讲自定义函数

创建（定义）函数 和 调用

PHP 函数准则：

- 函数的名称应该提示出它的功能
- 函数名称以字母或下划线开头（不能以数字开头）

```
function functionName() {
// 要执行的代码
}
//调用
functionName();
```

```
function test(){
    echo '我的名字是：写代码的猫叔';
}

test();

// 获取当前的时间
function nowTime(){
```

```
        echo date('Y-m-d H:i:s');
}

nowTime();
```

添加参数

为了给函数添加更多的功能，我们可以添加参数，参数类似变量

```
function myName($name){
        echo "我的名字是: $name";
}

myName("猫叔");
```

```
// 获取当前的时间
function nowTime($type){
    if($type=="time"){
        echo date('Y-m-d H:i:s');
    }
    else if($type=="day"){
        echo date('Y-m-d');
    }
}

nowTime("time");
```

多个参数

```
function plus($a,$b)
{
  echo $a+$b;
}

plus(2,3);
```

```
//严格模式
declare(strict_types=1);
function plus(int $a,int $b)
{
  echo $a+$b;
}
```

```
plus('2','13');
```

返回值

return ， 可有可无，但一般都有

```
function plus($a,$b)
{
  return $a+$b;
}

echo plus(2,3);
```

变量作用域

讲完了函数，就可以学习前面提到过的变量作用域了

变量的作用域是脚本中变量可被引用/使用的部分。

局部 和 全局(global) 作用域

- 在所有函数外部定义的变量，拥有全局作用域
- 除了函数外，全局变量可以被脚本中的任何部分访问
- 要在一个函数中访问一个全局变量，需要使用 global 关键字
- 函数内部声明的变量是局部变量，仅能在函数内部访问

```
$maoshu="写代码的猫叔";
function test(){
        global $maoshu;
        $age="35";
        echo $maoshu;
}
test();
echo $age;
```

static 作用域

当一个函数完成时，它的所有变量通常都会被删除。然而，有时候希望某个局部变量不要被删除。
要做到这一点，请在第一次声明变量时使用 **static** 关键字

每次调用该函数时，该变量将会保留着函数前一次被调用时的值，但是该变量仍然是函数的局部变量

```
function myTest()
{
    static $a=0;
    echo $a;
    $a++;
    echo PHP_EOL;    // 换行符
}

myTest();
myTest();
myTest();
```

函数的参数作用域

参数是通过调用代码将值传递给函数的局部变量

```
function test($maoshu){
        echo $maoshu;
}

test('猫叔');
var_dump($maoshu);
```

unset和isset函数

unset()可以删除变量，isset() 可以判断变量是否存在

```
$maoshu="写代码的猫叔";

function test(){
        // global $maoshu;
        $age="35";
    var_dump(isset($age));
        unset($age);

    var_dump(isset($age));
}
test();
```

```php
var_dump($maoshu);
```

超级全局变量

```
- $GLOBALS
- $_SERVER

- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION
```

$GLOBALS

是一个包含了全部变量的全局组合数组。变量的名字就是数组的键。

```php
print_r($GLOBALS);

$maoshu="猫叔";
$age=35;
function test3(){
        global $maoshu,$age;
        $age = "38";
    $maoshu = "写代码的猫叔";
}
test3();
echo $age;
echo $maoshu;
```

```php
$maoshu="猫叔";
$age=35;

function test2(){
    $GLOBALS["age"]="38";
    $GLOBALS["maoshu"] = "写代码的猫叔";

    echo $GLOBALS["maoshu"].'---'.$GLOBALS["age"];
}
```

```
test2();
echo PHP_EOL;
echo $age;
echo $maoshu;
```

$_SERVER

- `$_SERVER` 是一个包含了诸如头信息(header)、路径(path)、以及脚本位置(script locations)等等信息的数组。
- 这个数组中的项目由 Web 服务器创建。
- 不能保证每个服务器都提供全部项目；服务器可能会忽略一些，或者提供一些没有在这里列举出来的项目。

```
print_r($_SERVER)
```

资料来自 PHP 超级全局变量 | 菜鸟教程 (runoob.com)

元素/代码	描述
$_SERVER['PHP_SELF']	当前执行脚本的文件名，与 document root 有关。例如，在地址为 http://example.com/test.php/foo.bar 的脚本中使用 $_SERVER['PHP_SELF'] 将得到 /test.php/foo.bar。__FILE__ 常量包含当前(例如包含)文件的完整路径和文件名。 从 PHP 4.3.0 版本开始，如果 PHP 以命令行模式运行，这个变量将包含脚本名。之前的版本该变量不可用。
$_SERVER['GATEWAY_INTERFACE']	服务器使用的 CGI 规范的版本；例如，"CGI/1.1"。
$_SERVER['SERVER_ADDR']	当前运行脚本所在的服务器的 IP 地址。
$_SERVER['SERVER_NAME']	当前运行脚本所在的服务器的主机名。如果脚本运行于虚拟主机中，该名称是由那个虚拟主机所设置的值决定。(如: www.runoob.com)
$_SERVER['SERVER_SOFTWARE']	服务器标识字符串，在响应请求时的头信息中给出。 (如：Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	请求页面时通信协议的名称和版本。例如，"HTTP/1.0"。
$_SERVER['REQUEST_METHOD']	访问页面使用的请求方法；例如，"GET", "HEAD"，"POST"，"PUT"。
$_SERVER['REQUEST_TIME']	请求开始时的时间戳。从 PHP 5.1.0 起可用。 (如：1377687496)
$_SERVER['QUERY_STRING']	query string（查询字符串），如果有的话，通过它进行页面访问。
$_SERVER['HTTP_ACCEPT']	当前请求头中 Accept: 项的内容，如果存在的话。
$_SERVER['HTTP_ACCEPT_CHARSET']	当前请求头中 Accept-Charset: 项的内容，如果存在的话。例如："iso-8859-1,*,utf-8"。

$_SERVER['HTTP_HOST']	当前请求头中 Host: 项的内容，如果存在的话。
$_SERVER['HTTP_REFERER']	引导用户代理到当前页的前一页的地址（如果存在）。由 user agent 设置决定。并不是所有的用户代理都会设置该项，有的还提供了修改 HTTP_REFERER 的功能。简言之，该值并不可信。）
$_SERVER['HTTPS']	如果脚本是通过 HTTPS 协议被访问，则被设为一个非空的值。
$_SERVER['REMOTE_ADDR']	浏览当前页面的用户的 IP 地址。
$_SERVER['REMOTE_HOST']	浏览当前页面的用户的主机名。DNS 反向解析不依赖于用户的 REMOTE_ADDR。
$_SERVER['REMOTE_PORT']	用户机器上连接到 Web 服务器所使用的端口号。
$_SERVER['SCRIPT_FILENAME']	当前执行脚本的绝对路径。
$_SERVER['SERVER_ADMIN']	该值指明了 Apache 服务器配置文件中的 SERVER_ADMIN 参数。如果脚本运行在一个虚拟主机上，则该值是那个虚拟主机的值。(如：someone@runoob.com)
$_SERVER['SERVER_PORT']	Web 服务器使用的端口。默认值为 "80"。如果使用 SSL 安全连接，则这个值为用户设置的 HTTP 端口。
$_SERVER['SERVER_SIGNATURE']	包含了服务器版本和虚拟主机名的字符串。
$_SERVER['PATH_TRANSLATED']	当前脚本所在文件系统（非文档根目录）的基本路径。这是在服务器进行虚拟到真实路径的映像后的结果。
$_SERVER['SCRIPT_NAME']	包含当前脚本的路径。这在页面需要指向自己时非常有用。__FILE__ 常量包含当前脚本(例如包含文件)的完整路径和文件名。
$_SERVER['SCRIPT_URI']	URI 用来指定要访问的页面。例如 "/index.html"。

初级实战3 - 函数的使用

- 编写一个函数getFactors，接受一个正整数参数，计算并返回该整数的所有因子（除了1和自身的所有正整数因子）。
- 注：因子是能够整除给定数的正整数（例如，对于数值 24，它的因子包括 1、2、3、4、6、8、12 和 24。因为这些数都可以整除 24，没有余数。）
- 补充知识：怎么往数组内插入数据
- 要求使用函数

其他运算符

数组运算符

- +号运算符（合并）
  注意：只会保留第一个数组中的键值对，而忽略后面数组中相同键名的元素，如果想要合并

两个数组并覆盖相同键名的元素，可以使用array_merge()函数

```php
$a = [
  "m" => "测试",
  "n" => "猫叔"
];

$b = [
  "m" => "测试2",
  "o"=>"测试o",
  "p"=>"测试p",
  "q"=>[
      "cs"=>"二维测试"
  ]
];

$c=[
  "m" => "测试3",
  "cc"=>123
];

print_r($a+$b+$c);
```

- ==
- ===
  有相同的键/值对，且顺序相同、类型相同

```php
  $a = [1,2,3];
  $b = [1,2,"3"];
      var_dump($a == $b);
      var_dump($a === $b);

  $a = [
      "a"=>123,
      "b"=>456,
  ];

  $b = [
      "b"=>456,
      "a"=>123,
  ];

  var_dump($a == $b);
  var_dump($a === $b);
```

- != 和 <>
- !==

三元运算符 ?:

```php
// $name = $username?$username:"测试";

$name = isset($username)?$username:"测试";

// $name = $username?:"测试";
var_dump($name);
```

空合并运算符 ??

用于简化处理可能为null的变量或数组元素的情况。
它的作用是判断一个变量是否未定义或者为null，如果未定义或为null，则返回指定的默认值；否则返回该变量的值

```php
$name = $username ?? "Maoshu";
echo $name;
```

组合比较符 <=>

可比较整型 浮点型 字符串

```php
$c = $a <=> $b;
```

如果 $a > $b, 则 $c 的值为 1
如果 $a == $b, 则 $c 的值为 0
如果 $a < $b, 则 $c 的值为 -1

```php
$a = "acd";
$b = "ace";

var_dump($a <=> $b);
```

比较

数据类型的比较

虽然 PHP 是弱类型语言，但也需要明白变量类型及它们的意义
因为我们经常需要对 PHP 变量进行比较，包含松散和严格比较。

- 松散比较：使用两个等号 == 比较，只比较值，不比较类型。
- 严格比较：用三个等号（全等） === 比较，除了比较值，也比较类型。

例如，"42" 是一个字符串而 42 是一个整数。false 是一个布尔值而 "false" 是一个字符串。

```php
<?php
if(42 == "42") {
    echo '1、值相等';
}

echo PHP_EOL; // 换行符

if(42 === "42") {
    echo '2、类型相等';
} else {
    echo '3、类型不相等';
}
?>
```

问题：如果只想比较类型是否相同，怎么比较?

比较undefined、 0、false、null、空值

分析 isset() is_null() empty()的区别

```php
var_dump(!isset($name));
var_dump(is_null($name));
```

```php
$name = null;
var_dump(!isset($name));
var_dump(is_null($name));
var_dump(empty($name));
```

empty

注意：empty() 函数对于未定义的变量也会返回true，因此在使用empty() 函数之前，应该确保变量已经被定义

```php
$var1 = ""; // 空字符串
$var2 = 0; // 整数0
$var3 = null; // null
$var4 = false; // false
$var5 = array(); // 空数组 []
```

```php
var_dump(empty($var1));
var_dump(empty($var2));
var_dump(empty($var3));
var_dump(empty($var4));
var_dump(empty($var5));
```

字符串相关函数

```php
$strZh = "我是写代码的猫叔";

$strEn = "Hello MaoShu,I am a person who loves to share";
```

- strlen() 获取字符串长度
- strpos() 在字符串内查找一个字符或一段指定的文本，返回第一次出现的位置或false
- stripos() 同上，但不区分大小写
- strrpos() 同上上，返回最后一次出现的位置或false
- strripos() 同上，但不区分大小写
- explode() 把字符串打散成数组
- implode() 把数组拼接成字符串
- strtoupper() 把字符串转换为大写
- strtolower() 把字符串转换为小写
- ucfirst() 将单词的首字母转换为大写
- lcfirst() 将单词的首字母转换为小写
- ucwords() 将字符串中每个单词的首字母转换为大写
- str_replace($search, $replace, $string) 将字符串中的某个子字符串替换为另一个字符串
- strrev()：将字符串反转
- trim()：去除字符串两端的空格
- substr() 截取字符串的一部分
- mb_substr() 截取字符串的一部分(中文) 需要安装扩展mbstring

初级实战4 - 字符串函数的使用

编写一个函数，接受一个英文字符串参数"Hello MaoShu, HOW Are You?"，将字符串中的每个单词的首字母转换为小写，并返回修改后的字符串。

- 考察字符串的函数使用
- 考察数组的使用
- 考察循环的使用

```php
$str = 'Hello MaoShu,HOW,Are,You';

$arr = explode(' ',$str);
//   print_r($arr);

$newArr = [];
foreach ($arr as $key => $value) {
  if(strpos($value,',') !== false){
      $valueArr = explode(',',$value);
      $newValueArr = [];
      foreach ($valueArr as $kk => $vv) {
          $newValueArr[] = lcfirst($vv);
      }

      $newArr[] = implode(',',$newValueArr);
  }
  else{

      $newArr[]= lcfirst($value);
  }
}

echo implode(' ',$newArr);
```

数组相关函数

```php
$array = ["a","b","c","d"];
```

1. array()：创建一个数组。
2. count()：返回数组中元素的数量。
3. array_push($array, $newElement)：将一个或多个元素添加到数组的末尾。
4. array_unshift($array, $newElement)：将一个或多个元素添加到数组的开头。
5. array_pop()：删除并返回数组中的最后一个元素。
6. array_shift()：删除并返回数组中的第一个元素。
7. array_slice()：从数组中提取一部分元素，组成新的数组。
8. array_merge()：合并两个或多个数组。
9. array_reverse()：反转数组中的元素的顺序。
10. in_array()：检查数组中是否存在某个值。
11. array_key_exists()：检查数组中是否存在某个键。
12. array_keys()：返回数组中的所有键，组成新数组。
13. array_values()：返回数组中的所有值，组成新数组。

14. array_search()：在数组中搜索给定的值，并返回对应的键。
15. array_unique()：移除数组中的重复值（原数组不变）。
16. max() min() 最大值和最小值
17. sort() 数组排序（升序）
18. rsort() 数组排序（降序）
19. array_sum() 数组求和
20. array_product() 数组求乘积

暂时就想到这些函数，后面如果我们遇到了会再做补充。

初级实战5 - 数组函数的使用

编写一个PHP函数 `testArray($numbers)` ，该函数接收一个整数数组 `$numbers` 作为参数，并返回一个关联数组，包含以下统计信息：

- "count"： 数组中的元素个数。
- "sum"： 数组中所有元素的总和。
- "average"： 数组中元素的平均值。
- "max"： 数组中的最大值。
- "min"： 数组中的最小值。
- "sorted"： 按照从小到大排序的数组。

测试数组：

```
$numbers = [5, 9, 3, 2, 7, 1, 6];
```

时间、日期相关知识

函数

- time() 获取当前时间戳（10位），例如：1697520502
- microtime(true) 返回一个浮点数时间戳（秒数和微秒数的总和）
- date(格式，时间戳) 日期格式化

```
$time = time();
echo date('Y-m-d H:i:s',$time);
```

- strtotime(string)

```php
strtotime("next Monday");

$baseTime = strtotime("2023-10-01");
echo strtotime("next Monday", $baseTime);

$baseTime = time();
echo strtotime("+1 day", $baseTime);
```

- mktime($hour, $minute, $second, $month, $day, $year) 生成时间戳
- `date_create()` 来创建一个日期时间对象 `date_create('2023-11-01')`
- `date_format()` 用于将日期和时间格式化为指定的字符串格式

```php
date_format($date, "Y-m-d H:i:s");
date_format($date, "Y年m月d日 H:i:s")
```

- date_diff() 计算两个日期之间的差

```php
$date1 = date_create('2023-10-20 12:00:00');
$date2 = date_create('2023-10-21 12:00:00');
$diff = date_diff($date2, $date1);
echo $diff->format('%a 天');
echo $diff->format('%m 月');
echo $diff->format('%y 年 %m 月 %d 天');

-  %Y : 完整年份的差异
-  %y : 年份的差异（两位数）
-  %m : 月份的差异
-  %d : 天数的差异
-  %a : 总共的天数差异
-  %H : 小时的差异
-  %h : 小时的差异（12小时制）
-  %I : 分钟的差异
-  %S : 秒数的差异
-  %R : 正负
-  %r : 正负(必须是负的才会显示)
```

- `strftime($format, $timestamp)` ：根据指定的格式，将时间戳格式化为可读的日期和时间字符串，支持本地化的日期和时间格式 (php8已废弃)。

```php
$timestamp = time();
$dateString = strftime("%A %Y-%m-%d %H:%M:%S", $timestamp);
echo $dateString;
```

- `gmdate($format, $timestamp)` ：根据指定的格式，将GMT时间戳格式化为可读的日期和时间字符串。
  `gmdate()` 和 `date()` 的区别
  `gmdate()` 函数使用格林威治标准时间（GMT）作为默认的日期/时间基准，会忽略服务器的时区设置，始终返回格林威治标准时间（GMT）的日期和时间；
  `date()` 函数则使用本地时间作为基准，它会根据当前服务器的时区设置来格式化日期和时间。
- `date_default_timezone_set($timezone)` ：设置默认的时区。

"UTC", "localtime"本地时区, "Asia/Shanghai"上海

注意：只能在脚本开始时设置默认时区，不能在运行时动态设置。

- `timezone_identifiers_list()` ：返回所有可用时区标识符的数组。

DateTime对象

```php
$dateTime = new DateTime("2023-11-01 12:34:56");
//增加或减少指定的时间间隔
$dateTime->modify('+1 day');
$dateTime->modify('-1 hour');

//设置日期部分 setDate($year, $month, $day)
$dateTime->setDate(2024, 1, 3);

//设置时间部分 setTime($hour, $minute, $second)
$dateTime->setTime(13, 1, 5);

// 设置时区
$dateTime->setTimezone(new DateTimeZone("Asia/Shanghai"));
// 获取时区
$dateTimeZone = $dateTime->getTimezone();
print_r($dateTimeZone->getName());
print_r($dateTimeZone->getLocation());

//将日期时间格式化为指定的字符串格式
echo $dateTime->format("Y-m-d H:i:s");

//获取时间戳
$timestamp = $dateTime->getTimestamp();
echo $timestamp;

//计算时间差
$dateTime2 = new DateTime('2023-12-05');
```

```php
$diff = $dateTime->diff($dateTime2);
echo $diff->format('%R %m %d %h %i %s');
```

```php
$datetime = new DateTime('2024-01-01');
$interval = new DateInterval('P1D'); // 一天的时间间隔
$datetime->add($interval);
//$datetime->sub($interval);

echo $datetime->format('Y-m-d');
```

特殊的变量写法和常量

交流社区地址：https://maoshu.fun

特殊的变量写法

```php
$a ='maoshu';
$$a = "测试";

echo $a;
echo ' ---- ';
echo $$a;
echo ' ---- ';
echo $maoshu;
```

实际使用场景：

```php
for ($i=0; $i < 5; $i++) {
    $a = "maoshu".$i;
    $$a = "测试-".$i;
}

echo $maoshu1;
echo $maoshu2;
```

常量

常量值被定义后，在脚本的其他任何地方都不能被改变。
命名规则和变量类似，严格区分大小写，但无需$符
注意： 默认是全局的，可以在整个运行的脚本的任何地方使用。

```php
define("MAOSHU", "写代码的猫叔 maoshu.fun");
echo MAOSHU;

echo '<hr>';

const MAOSHU1 = "猫大叔";
echo MAOSHU1;

$a = 2;
define("MAOSHU".$a, "猫叔1");
echo constant("MAOSHU".$a);

echo '<hr>';
```

实际使用场景：

```php
for ($i=0; $i < 5; $i++) {
    $a = "MAOSHU".$i;
    // $$a = "测试-".$i;
    define($a,'测试-'.$i);
}

echo MAOSHU1;
echo MAOSHU2;
```

const和define的区别

- const不能在条件语句中定义常量

```php
$a = 1;
if($a==1){
    define("MAOSHU".$a, "猫叔1");
    echo constant("MAOSHU".$a);

    const MAOSHU2 = "猫大叔";
    echo MAOSHU2;
}
```

- const用于类成员变量的定义，define不可以用于类成员变量的定义，可用于全局常量。
- const可在类中使用，define不能

```php
class ceshi {

    const Ceshi = '猫叔';
    public function cc()
    {
        define('MS','写代码的猫叔');
        echo self::Ceshi;
    }

}
$c = new ceshi;
$c->cc();

echo '----';
echo MS;
```

常量和变量的区别：

- 常量前面没有美元符号($)，而且不能有
- 常量可以不用理会变量的作用域在任何地方定义和访问
- 常量一旦定义就不能重新定义或取消定义
- 常量的值只能是标量（字符串、整数、浮点数、布尔值）,注意：现在也支持数组了

获取所有的常量

```php
get_defined_constants()
get_defined_constants(true)
get_defined_constants(true)["user"]
```

魔术常量

魔术常量：它的值随着它在代码中的位置改变而改变

`__LINE__`

文件中的当前行号

```php
echo '这是第 " ' . __LINE__ . ' " 行';
```

`__FILE__`

文件的完整路径和文件名，包含（盘符）根目录

```
echo '该文件的完整路径为 " ' . __FILE__ . ' " ';
```

`__DIR__`

文件所在的目录

```
echo '该文件位于 " ' . __DIR__ . ' " ';
```

`__FUNCTION__`

该函数被定义时的名字（区分大小写）

```
function test() {
    echo  '函数名为: ' . __FUNCTION__ ;
}
test();
```

`__CLASS__`

该类被定义时的名字（区分大小写）

```
<?php
class testClass {
    function test() {
        echo '类名为: '  . __CLASS__ . "<br>";
        echo  '函数名为: ' . __FUNCTION__ ;
    }
}
$t = new testClass();
$t->test();
?>
```

`__NAMESPACE__`

命名空间

```
namespace MyProject;
echo '命名空间为: "', __NAMESPACE__, '"'; // 输出 "MyProject"
```

`__METHOD__`

包含了：命名空间 类名 函数名

```php
namespace MAOSHU;
class MyClass {
    public function myMethod() {
        echo __METHOD__; // 输出: MAOSHU\MyClass::myMethod
    }
}

$obj = new MyClass();
$obj->myMethod();
```

`__TRAIT__`

当前使用的 trait 的名称

```php
trait MyTrait {
    public function myMethod() {
        echo "trait的名称为: " . __TRAIT__;
    }
}

class MyClass {
    use MyTrait;
}

$obj = new MyClass();
$obj->myMethod();
```

包含文件

在当前文件内引入其他PHP文件、HTML文件或文本文件等，一般用于包含公共方法，公共页面等，例如header footer sider等网页通用部分。

include 和 require 语句

区别：
include 和 require 除了处理错误的方式不同之外，在其他方面都是相同的：

- require 生成一个致命错误（E_COMPILE_ERROR），在错误发生后脚本会停止执行。
- include 生成一个警告（E_WARNING），在错误发生后脚本会继续执行。

因此：
如果你希望被包含的文件是必需的且缺少文件会导致脚本无法正常运行，应使用require（推

荐）。
如果你希望被包含的文件是可选的，或者即使缺少文件也希望脚本继续执行，可以使用include。

例如：

header.php

```
<?php
        $siteTitle = "猫叔星球(maoshu.fun)";
?>
<html>
<head>
<meta charset="utf-8">
<title><?php echo $siteTitle;?></title>
</head>
```

index.php

```
<?php include 'header.php'; ?>
<body>
<h1>欢迎来到<?php echo $siteTitle;?> !</h1>
<p>有问必答。</p>

</body>
</html>
```

include_once 和 require_once 语句

和上面的一样，但他只会调用一次，防止重复调用

面向对象（OO）

概念

对象：

在现实世界里我们所面对的事情都是对象，如苹果 猫 电视机等。
而在面向对象的程序设计(OOP)中，对象是一个由信息及对信息进行处理的描述所组成的整体，是对现实世界的抽象。

例如：
小猫 名字：Tom，性别：公，花色：橘色；会叫，会吃饭，会打闹

小狗 名字：Jack，性别：公，花色：黑色；会叫，会吃饭，会打闹

区分：谁在叫，谁在吃饭，谁在打闹

面向对象：

- 面向对象是一种编程思想和方法，它将程序中的**数据**和**操作数据**的方法封装在一起，形成"对象"，并通过对象之间的交互和消息传递来完成程序的功能。
- 面向对象编程(OOP)强调数据的**封装、继承、多态**和**动态绑定**等特性，使得程序具有更好的可扩展性、可维护性和可重用性。

对象的主要三个特性：

- 对象的行为：对象可以执行的操作，比如：开灯，关灯就是行为。
- 对象的形态：对对象不同的行为是如何响应的，比如：颜色，尺寸，外型。
- 对象的表示：对象的表示就相当于身份证，具体区分在相同的行为与状态下有什么不同（在面向对象编程中，对象的表示通常通过类来实现）。

面向对象编程的三个主要特性：

- 封装：指将对象的属性和方法封装在一起，使得外部无法直接访问和修改对象的内部状态。通过使用访问控制修饰符（public、private、protected）来限制属性和方法的访问权限，从而实现封装。
- 继承：指可以创建一个新的类，该类继承（extends）了父类的属性和方法，并且可以添加自己的属性和方法。通过继承，可以避免重复编写相似的代码，并且可以实现代码的重用。
- 多态：指可以使用一个父类类型的变量来引用不同子类类型的对象，从而实现对不同对象的统一操作。多态可以使得代码更加灵活，具有更好的可扩展性和可维护性。在 PHP 中，多态可以通过实现接口（interface）和使用抽象类（abstract class）来实现。

类（class）

- 定义了一件事物的抽象特点。
- 类的定义包含了数据的形式以及对数据的操作。

类的定义和调用

类的定义

```
class Animal {
        public $name = "小猫仔";
    public function eat() {
        echo " 在吃饭.";
```

```
    }
}
```

类的调用

new 实例化对象

```
$cat = new Animal;
//$cat = new Animal();
echo $cat->name;
$cat->eat();
```

方法和属性

类方法（函数）

```
class Animal {
        public $name = "小猫仔";
    public function eat() {
        echo " 在吃饭.";
    }
    function say() {
        echo " 在说话.";
    }
}
```

$this

代表自身的对象

```
class Animal {
        public $name = "小猫仔";
    public function eat() {
        echo $this->name." 在吃饭.";
    }
    function say() {
        echo $this->name." 在说话.";
    }
}

$cat = new Animal;
$cat->name = "小花";
$cat->say();
```

```php
$dog = new Animal;
$dog->name = "小黑";
$dog->say();
```

访问控制

关键字 public、private、protected

- **public（公有）**：公有的类成员可以在任何地方被访问。
- **protected（受保护）**：受保护的类成员则可以被其自身以及其子类和父类访问（咱家的）。
- **private（私有）**：私有的类成员则只能被其定义所在的类访问（自己的）。

`__construct` 构造函数

- 构造函数是一种特殊的方法，在创建一个新对象时，它会被自动调用。
- 它可以用来 **初始化** 对象的属性或执行其他必要的操作
- 没有返回值

```php
  class Animal {
        private $name;
        private $birth;
        private $age;

    public function __construct($name,$birth)
    {
        $this->name = $name;
        $this->birth = $birth;

        $days = (time() - strtotime($this->birth))/3600/24;
        $this->age = floor($days);
    }

    public function eat() {
        echo  $this->name." 在吃饭.";
    }
}

$cat = new Animal("猫仔","2023-05-23");
echo $cat->name;
echo ' -- ';
echo $cat->birth;
echo ' -- ';
echo $cat->age;
```

`__destruct` 析构函数

- 析构函数是一种特殊的方法，它在对象被销毁时自动调用
- 它可以用来执行一些清理操作，例如释放资源或关闭数据库连接。
- 当对象不再被引用或脚本执行结束时，析构函数会被自动调用。

```
class MyClass {
    public function say($i)
    {
      echo 'saying-'.$i;
    }
    public function __destruct() {
        echo "析构函数被调用\n";
    }
}

// 创建对象
$obj = new MyClass();

// 执行其他操作...
for ($i=0; $i < 4; $i++) {
    if($i==3){
        unset($obj);
    }
    if($obj)
        $obj->say($i);
}
```

static 静态变量 和 self

「静态」指的是无需对类进行实例化，就可以直接调用这些属性和方法
所有对静态变量进行的操作都会对所有对象起作用
举例：小猫小狗听到指令来吃饭，指令变化，全部都要听从

```
public static $cat = "猫叔";

echo self::$cat;
```

类常量

静态属性与类常量相似，唯一的区分是类常量不可以更改，静态属性可以更改
使用场景：所有的对象共用一个属性

```
const MAOSHU = "猫叔";

echo self::MAOSHU;
```

static 静态方法

```
public static function say() {
        echo self::$name;
}
```

可以调用静态方法、静态变量
可以调用非静态方法、非静态变量

```
public function eat() {

        echo $this->name." 在吃饭.";

}
public static function say() {

        echo (new self)->eat();

}
```

类的继承（extends）：

指可以创建一个新的类，该类继承（extends）了父类的属性和方法，并且可以添加自己的属性和方法。通过继承，可以避免重复编写相似的代码，并且可以实现代码的重用。

注意：继承不一定能访问

```
class Animal {
    public $name="小动物";
    protected $age=3;
    private $birth='2023';
}

class Cat extends Animal {}


var_dump(new Animal);
var_dump(new Cat);
```

```php
class Animal {
    protected $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function eat() {
        echo $this->name . " 在吃饭.";
    }
}

class Cat extends Animal {
    public function meow() {
        echo $this->name . " 在喵呜.";
    }
}

$cat = new Cat("Tom");
var_dump($cat);
$cat->eat();  // 继承自父类 Animal 的方法
$cat->meow();  // 子类 Cat 自己的方法
```

方法和属性重写

如果从父类继承的方法或属性不能满足子类的需求，可以对其进行改写

final 关键字

作用：

- 防止类被继承
- 防止类的方法被重写

如果在一个类前加final，那么这个类就不能被继承；

```php
final class myClass {
    // 类的内容
}
```

如果在一个方法前加final，那么这个方法就不能被重写

```php
    final public function eat() {
        echo $this->name . " 在吃饭.";
```

```
    }
```

注意：final不能用于属性

调用父类方法

parent::
parent::__construct()

静态延迟绑定 static

是指在运行时根据实际调用的类来确定静态方法或属性的绑定
语法：`static::$name`

```
class Animal {
    protected static $name="小动物";

    public static function eat() {
        echo self::$name. " 在吃饭.";
        echo ' ---- ';
        echo static::$name. " 在吃饭.";
    }
}

class Cat extends Animal {
    protected static $name="小猫";
}

Animal::eat();
Cat::eat();
```

类的多态

- 多态性允许不同类的对象对相同的消息作出不同的响应。
- 多态性通过方法重写（覆盖）和方法重载来实现。
- 方法重写是指子类重写父类的方法，以改变方法的实现细节。
- 方法重载是指在同一个类中根据参数个数或类型不同来实现不同功能。
- 需要注意的是，多态性只适用于继承关系的类。子类必须重写父类的方法才能实现多态性。

```
class Animal {
    protected $name="动物";
    public function makeSound() {
```

```
        echo "$this->name 在吼叫。";
    }
}

class Dog extends Animal {
    protected $name="小狗";
    public function makeSound() {
        echo "$this->name 在汪汪。";
    }
}

class Cat extends Animal {
    protected $name="小猫";
    public function makeSound() {
        echo "$this->name 在喵喵。";
    }
}

$animal = new Animal();
$dog = new Dog();
$cat = new Cat();

$animal->makeSound();
$dog->makeSound();
$cat->makeSound();
```

方法重载

```
$args = func_get_args();
$numArgs = func_num_args();

function test(){

    $args = func_get_args();
    $numArgs = func_num_args();

    var_dump($args[1]);
    var_dump($numArgs);
}

test(1,2,3,4);
```

```php
class Animal {
    public function makeSound() {
        echo "动物在吼叫";
    }
}


class Cat extends Animal {
    public function makeSound() {
        // $args = func_get_args();
        $numArgs = func_num_args();

        switch($numArgs){
            case 2:
                echo '执行参数个数为2的事件';
                break;
            case 3:
                echo '执行参数个数为3的事件';
                break;
            default:
                echo '执行默认事件';
        }
    }
}

// $animal = new Animal();
$cat = new Cat();

$cat->makeSound("测试","2");
```

接口和抽象类

interface(接口)

```php
interface Animals {
        public function a();
        public function b();
}

class Cat implements Animals {
        public function a(){

        };
        public function b(){
```

```
        };
}
```

接口是指一组方法的集合，不是类，不能被实例化。

- 可以指定某个类必须**实现**哪些方法，但不需要定义这些方法的具体内容
- 只可以使用public
- 通常用于定义一些规范，让代码更加有条理 不易出错。
  例如：小动物必须要吃饭和睡觉，否则就会死！这是必须的，每个小动物都必须有这2个方法！

```php
interface Animals {
    const MAOSHU = "猫叔";
    public function eat();
    public function sleep($hours);
    public static function jump();
}

class Cat implements Animals {
    public function eat() {
        echo "Cat 在吃饭.";
    }

    public function sleep($hours) {
        echo "Cat 要睡 $hours 小时.";
    }

    public static function jump(){
        echo '跳跳跳';
    }
}

$cat = new Cat();
$cat->eat();
$cat->sleep(18);

echo Cat::MAOSHU;
echo Cat::jump();
```

```php
interface Animals {
    public function eat();
    public function sleep($hours);
}
```

```php
interface Sports {
    public function run();
    public function jump();
}
class Cat implements Animals,Sports {
    public function eat() {
        echo "Cat 在吃饭";
    }

    public function sleep($hours) {
        echo "Cat 要睡 $hours 小时";
    }

    public function run() {
        echo "Cat 在跑步";
    }

    public function jump() {
        echo "Cat 在蹦跳";
    }
}
```

抽象类和抽象方法

和接口非常类似，使用它也是定义一种约束或规范，适合较大型的项目或库使用

抽象类

```php
abstract class Animals{

}
```

- 抽象类是一种特殊的类，只能被继承，不能被实例化
- 抽象类用于定义一组相关的方法，但这些方法的具体实现由继承它的子类来完成。
- 子类继承抽象类后，必须实现抽象类中的所有抽象方法。
- 抽象类可以包含抽象方法和普通方法

抽象方法

```php
abstract public function xxx();
abstract protected function xxx();
```

- 抽象方法是没有具体实现的方法，只有方法的声明，而不需要方法体。

- 抽象方法只能存在于抽象类中。
- 可以使用protected，但不能使用private私有。

```php
abstract class Animals
{
    abstract public function eat();
    abstract protected function sleep($hours);

    public function play()
    {
      echo '玩耍';
    }
}

class Cat extends Animals
{
    public function eat()
    {
        echo "Cat 在吃饭.";
    }

    protected function sleep($hours)
    {
        echo "Cat 要睡 $hours 小时.";
    }
}


$cat = new Cat();
$cat->eat();
//$cat->sleep(12);
$cat->play();
```

抽象类与接口的区别

1. 抽象类可以包含非抽象方法的实现，而接口只能包含方法的声明，没有方法的实现。
2. 类只能继承一个抽象类，但可以实现多个接口。
3. 抽象类可以有构造函数，而接口不能有构造函数。
4. 抽象类中的方法可以有public、protected和private访问修饰符，而接口中的方法只能是public。
5. 子类继承抽象类时，必须实现抽象类中的所有抽象方法，否则子类也必须声明为抽象类。子类实现接口时，必须实现接口中的所有方法。

trait代码复用

- 解决类的单一继承问题
- 可同时使用多个trait，用逗号隔开
- 把常用的、通用的代码抽离出来，写成trait

```
trait A{

}
trait B{

}
class C {
        use A,B;
}
```

和类的继承非常像，但是trait里面不能有类常量，且trait不能被实例化。
根据下面类的继承来修改，对比看下他们的相同点：

```
class Animal {
    protected $name;
    const MAOSHU="猫叔";

    public function __construct($name) {
        $this->name = $name;
    }

    public function eat() {
        echo $this->name . " 在吃饭.";
    }
}

class Cat extends Animal {
    public function meow() {
        echo $this->name . " 在喵呜.";
    }
}

$cat = new Cat("Tom");
var_dump($cat);
$cat->eat();  // 继承自父类 Animal 的方法
$cat->meow();  // 子类 Cat 自己的方法
```

- trait中可使用抽象方法
- trait中可使用静态属性和静态方法
- trait中可使用其他trait
- trait中可使用parent

```php
class MainClass{
    public function main()
    {
      echo '这是主方法的'.__METHOD__;
    }
}
trait Animal {

    public function eat() {
        parent::main();
        echo $this->name . " 在吃饭.";
        echo __TRAIT__;
    }
}

class Cat extends MainClass {
    use Animal;
    protected $name;
    public function __construct($n) {
        $this->name = $n;
    }
    public function meow() {
        echo $this->name . " 在喵呜.";
        echo __TRAIT__;
    }
}

$cat = new Cat("Tom");
var_dump($cat);
$cat->eat();  // 继承自父类 Animal 的方法
echo '----';
$cat->meow();  // 子类 Cat 自己的方法
```

同名冲突

当一个类同时引入了多个Trait，并且这些Trait中存在同名方法时，就会产生方法冲突。

```
use A,B{
        //A::eat insteadof B;
    B::eat insteadof A;
    //别名定义
    A::eat as Aeat;
}
```

```
trait A
{
    public function eat(){
        echo '这是A的';
    }
}

trait B
{
    public function eat(){
        echo '这是B的';
    }
}

class T{
    use A,B{
        A::eat insteadof B;
        B::eat as Beat;
    }
}

$t = new T;

$t->eat();
$t->Beat();
```

表单和请求

主要用到了`$_GET`和`$_POST`超全局变量，分别对应两种不同的请求方式

表单

input select radio checkbox submit

```
<form action="./user.php" method="post">
    用户名: <input type="text" name="username">
    密码: <input type="text" name="password">
```

```
    <select multiple name="arr[]">
        <option value="">请选择:</option>
        <option value="1">大猫</option>
        <option value="2">二猫</option>
        <option value="3">三猫</option>
    </select>
    <input type="submit" value="提交">
</form>
```

关于网络请求

GET请求和POST请求是两种常用的HTTP请求方法，用于从客户端向服务器发送数据

GET请求：

通过URL（网址）参数将数据附加在URL上发送给服务器。这些参数以键值对的形式出现在URL的末尾，使用问号"?"将URL和参数分隔开。

POST请求：

将数据作为请求的主体发送给服务器，而不是附加在URL上。这使得POST请求更适合发送敏感数据或超长大段内容，例如表单中的密码、博客文本内容等。

区别：

- GET参数对任何人都是可见的，POST参数对任何人都是不可见的
- GET对发送信息的量也有限制，不适合大型的变量值，它的值不能超过 2000 个字符；POST对发送信息的量没有限制（默认POST 发送的最大值为 8 MB，但可通过设置 php.ini 的 post_max_size 进行更改；发送的变量数最大值max_input_vars）
- 可以在收藏夹中收藏GET请求页面，或者发送带参数的网址给别人可以直接访问，例如带页码的网址、带商品id的淘宝商品、带文章id的博文。这一点POST做不到。

超全局变量 `$_REQUEST`

> `$_REQUEST` 变量包含了 `$_GET` 、 `$_POST` 和 `$_COOKIE` 的内容
> `$_SERVER` 获取请求方式

htmlspecialchars()函数

用于将字符串中的特殊字符转换为HTML实体，以避免在HTML文档中引起解析错误或安全漏洞。

- `&` （和号） 成为 `&`

- `"` （双引号） 成为 `"`
- `'` （单引号） 成为 `'`
- `<` （小于） 成为 `<`
- `>` （大于） 成为 `>`

mysql数据库

本课程只讲解最简单的增删改查语句，不做更深入的语句讲解。

配置环境变量

`mysql -u root -p`

查看数据库列表：
`SHOW DATABASES;`

使用数据库
`USE xxx;`

查看数据表列表：
`SHOW TABLES;`

数据库工具：SQL_Front(小皮内置)、Navicat、DBeaver

> 操作数据库的方式：MySQLi 和 PDO

MySQLi

首先要启用扩展mysqli

面向过程

```php
<?php
$servername = "localhost";
$username = "root";
$password = "root";

// 创建连接
$conn = mysqli_connect($servername, $username, $password);

// 检测连接
if (!$conn) {
    die("连接失败: " . mysqli_connect_error());
}
echo "连接成功";
```

```php
//关闭
mysqli_close($conn);
?>
```

面向对象

```php
<?php
$servername = "localhost";
$username = "root";
$password = "root";

// 创建连接
$conn = new mysqli($servername, $username, $password);

// 检测连接
if ($conn->connect_error) {
    die("连接失败: " . $conn->connect_error);
}
echo "连接成功";

//关闭
$conn->close();
?>
```

resource 资源类型

- is_resource() 是否为资源类型
- get_resource_type() 获取资源的类型名称

```php
$servername = "localhost";
$username = "root";
$password = "root";
$conn = mysql_connect($servername, $username, $password);
var_dump(is_resource($conn));
var_dump(gettype($conn));
var_dump(get_resource_type($conn));
```

```php
$handle = fopen("test.txt", "r");
var_dump($handle);
var_dump(is_resource($handle));
var_dump(gettype($handle));
var_dump(get_resource_type($handle));
```

```
//$content = fread($handle, filesize("test.txt"));
//echo $content;
fclose($handle);
```

> 接下来将主讲面向对象方式操作

创建数据库

```
// 创建数据库
$sql = "CREATE DATABASE IF NOT EXISTS maoshuDB";
//if (mysqli_query($conn, $sql)) {
if ($conn->query($sql) === TRUE) {
    echo "数据库创建成功";
} else {
    echo "数据库创建失败 " . $conn->error;
}
```

创建数据库表

- NOT NULL - 每一行都必须含有值（不能为空），null 值是不允许的。
- DEFAULT value - 设置默认值
- UNSIGNED - 使用无符号数值类型，0 及正数
- AUTO INCREMENT - 设置 MySQL 字段的值在新增记录时每次自动增长 1
- PRIMARY KEY - 设置数据表中每条记录的唯一标识。 通常列的 PRIMARY KEY 设置为 ID 数值，与 AUTO_INCREMENT 一起使用。

每个表都应该有一个主键(本列为 "id" 列)，主键必须包含唯一的值。

```
//$conn->query("USE maoshuDB");
// 使用 sql 创建数据表
$sql = "CREATE TABLE IF NOT EXISTS user (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(30) NOT NULL COMMENT \"用户名\",
  password VARCHAR(64) NOT NULL COMMENT '密码',
  phone VARCHAR(11),
  status INT(1) NOT NULL DEFAULT 1,
  create_date TIMESTAMP
  ) COMMENT '用户'";
//if (mysqli_query($conn, $sql)) {
if ($conn->query($sql) === TRUE) {
    echo "数据表创建成功";
} else {
```

```
    echo "数据表创建错误: " . $conn->error;
}
```

> 注：如果要加注释，那么后面加上 COMMENT 即可

插入数据

> **注意：** 如果列设置 AUTO_INCREMENT (如 "id" 列) 或 TIMESTAMP (如 "create_date" 列)，我们就不需要在 SQL 语句中指定值； MySQL 会自动为该列添加值。

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

用`$conn->query($sql) === TRUE`来判断是否插入成功

```
$sql = "INSERT INTO user (username, password, phone)
VALUES ('da mao', '123456', 13888888888)";
//if (mysqli_query($conn, $sql)) {
if ($conn->query($sql) === TRUE) {
    echo "新记录插入成功";
} else {
    echo "错误: " . $sql . "<br>" . $conn->error;
}
```

字符串进行转义处理

`$conn->real_escape_string($string);`
可以确保特殊字符被正确转义，从而避免了SQL注入等安全问题，并保证查询语句的语法正确性，例如：

```
$name="这是一个'送命题'";
```

插入多条数据

```
$sql = "INSERT INTO user (username, password, phone)
VALUES
    ('大猫', '123456', 13888888888),
    ('小猫', '123456', 13888888887),
    ('中猫', '123456', 13888888886)
    ";
$conn->query($sql);
```

foreach循环插入，或者使用语句：
`mysqli_multi_query()`

```
$sql = "INSERT INTO user (username, password, phone)
VALUES ('大猫', '123456', 13888888888);";

$sql .= "INSERT INTO user (username, password, phone)
VALUES ('小猫', '123456', 13888888887);";

$sql.= "INSERT INTO user (username, password, phone)
VALUES ('中猫', '123456', 13888888886);";

//if (mysqli_multi_query($conn, $sql)) {
if ($conn->multi_query($sql) === TRUE) {
    echo "新记录插入成功";
} else {
    echo "错误: " . $sql . "<br>" . $conn->error;
}
```

查询数据

```
SELECT column1,column2 FROM table_name
```

```
$sql = "SELECT * FROM user";
$result = $conn->query($sql);
//if (mysqli_num_rows($result) > 0) {
if ($result->num_rows > 0) {
    echo "$result->num_rows 条结果";
    // 输出数据
    //while($row = mysqli_fetch_assoc($result)) {
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["username"]. " " .
$row["phone"]. "<br>";
    }
} else {
    echo "0 条结果";
}
```

- `$result->num_rows` 获取到的数据条数
- `$result->fetch_assoc()` 是一种用于从结果集中获取一行数据的函数。它以关联数组的形式返回结果，其中列名作为键，对应的值则是该行中各列的值
- 除了fetch_assoc 还有 fetch_row和fetch_array和fetch_all

```php
while($row = $result->fetch_row()) {
    echo "id: " . $row[0]. " - Name: " . $row[1]. " " . $row[2]. "<br>";
}
```

```php
    //$datas = mysqli_fetch_all($result, MYSQLI_ASSOC);
    $datas = $result->fetch_all(MYSQLI_ASSOC);
    var_dump($datas);
```

`free_result()`

释放结果集所占用的内存资源。在使用 `query()` 函数执行查询后，会返回一个结果集对象，该结果集对象占用一定的内存空间。

```php
//mysqli_free_result($result);
$result->free_result();
```

where 条件筛选

WHERE用于在查询中指定筛选条件，来限制返回的行

```
SELECT 列名
FROM 表名
WHERE 条件;
```

条件判断：

`> < = != <> >= <=`

`like "%xxx%"` `like "%xxx"` `like "xxx%"`

`between and`

`is null` `is not null`

`in (a,b,c)`

and or not 括号

order by 排序

ASC升序 、DESC降序

根据两列进行排序，用逗号隔开即可，注意：只有第一列的值相同时才使用第二列

offset limit

```
SELECT 列名
FROM 表名
LIMIT 行数 OFFSET 偏移量;
```

更新数据

```
UPDATE 表名
SET column1=value, column2=value2,...
WHERE 条件
```

注意：如果省去 WHERE 子句，所有的记录都会被更新

使用计算表达式更新字段

```
UPDATE 表名 SET 列名 = 列名 + 值 WHERE 条件;
```

```
$sql = "UPDATE user SET num = num*3 WHERE id = 2";
$result = $conn->query($sql);
```

删除数据

```
DELETE FROM 表名
WHERE 条件
```

注意：如果省去 WHERE 子句，所有的记录都会被删除

删除表中的重复行

```
DELETE t1 FROM 表名 t1, 表名 t2 WHERE t1.列名 = t2.列名 AND t1.id > t2.id;
```

修改数据表

```
修改数据表名：
ALTER TABLE 旧表名 RENAME 新表名;

重置自增起始值为1：
ALTER TABLE 表名 AUTO_INCREMENT = 1;

添加新的列：
ALTER TABLE 表名 ADD COLUMN 列名 数据类型;
```

```
修改列名和数据类型：
ALTER TABLE 表名 CHANGE COLUMN 旧列名 新列名 新数据类型;

修改列的默认值：
ALTER TABLE 表名 ALTER COLUMN 列名 SET DEFAULT 默认值;

删除列：
ALTER TABLE 表名 DROP COLUMN 列名;

修改列的注释：
ALTER TABLE 表名 MODIFY COLUMN 列名 数据类型 COMMENT '新注释';

删除数据表：
DROP TABLE IF EXISTS 表名

清空数据表：
TRUNCATE TABLE 表名;

删除数据库：
DROP database
```

咱们只讲了最基本的mysql，关于关联查询 索引 等更深入的知识就需要专门学习了。

预处理语句

预处理语句（Prepared Statements），用于执行带有参数的SQL语句。

- 预处理语句可以提高安全性，对于防止 SQL 注入是非常有用的
- 允许重复使用相同的sql模板而只需更改参数，提高执行效率。

关于sql注入

当涉及到SQL查询时，注入攻击是一种常见的安全漏洞

1. 直接执行用户输入的SQL查询：

```
$query = $_GET['query'];
$result = $conn->query($query);
```

如果用户能够直接执行他们自己构造的SQL查询，就会产生严重的安全问题。恶意用户可以执行任意的SQL语句，包括删除、修改或泄露敏感数据。

2. 使用字符串拼接的方式构建SQL查询：

```php
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM user WHERE username = '" . $username . "' AND password = '" .
$password . "'";
$result = $conn->query($sql);
```

这种方式存在安全风险，如果用户在输入框中输入恶意代码，可以破坏原始的SQL查询结构，例如输入 `' OR 1=1 --` ，会导致查询条件永远为真，从而绕过身份验证。

3. 转义特殊字符也不保险：

```php
$username = $conn->real_escape_string($_POST['username']);
$password = $conn->real_escape_string($_POST['password']);

$query = "SELECT * FROM users WHERE username = '" . $username . "' AND password =
'" . $password . "'";
$result = $conn->query($query);
```

我们使用 `->real_escape_string()` 函数对用户输入的用户名和密码进行转义，以防止SQL注入，但仍然可能存在注入漏洞。
假设用户在用户名输入框中输入 `admin'--` ，并在密码输入框中输入任意值。由于转义字符未正确处理，生成的SQL查询语句如下

```sql
SELECT * FROM users WHERE username = 'admin'--' AND password = 'password'
```

在这种情况下， `--` 是SQL注释的开始，后面的 `AND password = 'password'` 将被忽略。这样，攻击者可以绕过密码验证并成功登录，即使没有提供正确的密码。

预处理方法

查询

```php
$sql = "SELECT * FROM user WHERE username = ? AND password = ? ";
$stmt = $conn->prepare($sql);
//绑定
$stmt->bind_param("ss", $username, $password);
//执行
$stmt->execute();

//获取结果
$result = $stmt->get_result();
print_r('<pre>');
```

```
print_r($result->fetch_all());

//关闭
$stmt->close();
```

插入

```
// 预处理
$sql = "INSERT INTO user (username, password, phone)
VALUES (?, ?, ?)";
$stmt = $conn->prepare($sql);
//绑定
$stmt->bind_param("sss", $username, $password, $phone);
//执行
$stmt->execute();

//获取结果
if ($stmt->affected_rows > 0) {
    echo "插入成功";
} else {
    echo "插入失败";
}


//关闭
$stmt->close();
```

其中的`?`为占位符

通过告诉数据库参数的数据类型，可以降低 SQL 注入的风险。参数有以下四种类型:

- i - integer（整型）
- d - double（双精度浮点型）
- s - string（字符串）
- b - BLOB（binary large object:二进制大对象）

每个参数都需要指定类型。

php8新方法

```
$name="这是一个'送命题'";
//mysqli_execute_query
$sql = "INSERT INTO user (username, password, phone)
VALUES (?, ?, ?)";
```

```php
$conn->execute_query($sql,[$name,'123456','13888888888']);
$conn->execute_query($sql,[$name,'123456','13888888888']);
```

PDO

安装php_pdo_mysql扩展，用phpinfo() 函数来查看是否安装了PDO扩展

PDO执行sql语句分为2种方法：

exec() 方法：

- 适用于执行不返回结果集的SQL语句，例如INSERT、UPDATE和DELETE等操作。
- exec() 方法返回受影响的行数（即被插入、更新或删除的行数），如果没有受影响的行，返回0。
- 通常用于执行不需要获取结果集的简单操作。

query() 方法：

- 适用于执行返回结果集的SQL语句，例如SELECT查询。
- query() 方法返回一个PDOStatement对象，你可以使用该对象的方法（如 fetch() 、fetchAll() ）来获取查询结果。
- 通常用于执行需要获取结果集的查询操作。

创建连接

```php
<?php
$dbms='mysql';     //数据库类型
$host='localhost'; //数据库主机名
$dbName='maoshudb';    //使用的数据库
$user='root';      //数据库连接用户名
$pass='root';          //对应的密码
$dsn="$dbms:host=$host;dbname=$dbName";

try {
    $conn = new PDO($dsn, $user, $pass); //初始化一个PDO对象
    echo "连接成功<br/>";
        // 关闭连接
    $conn = null;
} catch (PDOException $e) {
    die ("错误!: " . $e->getMessage() . "<br/>");
}
?>
```

创建数据库

```
    //设置 PDO 错误模式为异常
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    //PDO::ERRMODE_WARNING 警告
    //PDO::ERRMODE_SILENT 错误码
    //$conn->errorCode()
    //$conn->errorInfo()

// 创建数据库
$sql = "CREATE DATABASE 数据库名";
$conn->exec($sql);
echo "数据库创建成功";
```

创建数据表

```
$sql = "CREATE TABLE IF NOT EXISTS user6 (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(30) NOT NULL COMMENT \"用户名\",
    password VARCHAR(64) NOT NULL COMMENT '密码',
    phone VARCHAR(11),
    status INT(1) NOT NULL DEFAULT 1,
    create_date TIMESTAMP
    ) COMMENT '用户'";
$conn->exec($sql);
```

查询数据

```
    // 执行查询语句
$sql = "SELECT * FROM user WHERE id<5";
$result = $conn->query($sql);

// 获取结果
$result = $result->fetchAll(PDO::FETCH_ASSOC);
var_dump($result);
```

更新数据

```
    $sql = "UPDATE user SET num = num+3 WHERE id =2";
$stmt = $conn->exec($sql);
var_dump($stmt);
```

删除数据

```
    $sql = "DELETE FROM user WHERE id =26";
$stmt = $conn->exec($sql);
```

插入数据

和mysqli的类似，不做特殊说明了
预处理

```
// 执行插入语句
$sql = "INSERT INTO 表名 (列1, 列2, 列3) VALUES (:value1, :value2, :value3)";
$stmt = $conn->prepare($sql);

// 绑定参数
$stmt->bindParam(':value1', $value1);
$stmt->bindParam(':value2', $value2);
$stmt->bindParam(':value3', $value3);

// 设置参数值
$value1 = "值1";
$value2 = "值2";
$value3 = "值3";

// 执行插入语句
$stmt->execute();

    // 获取插入的条数
$rowCount = $stmt->rowCount();
```

批量插入

```
$sql = "INSERT INTO user (username, password, phone) VALUES (?,?,?)";
$stmt = $conn->prepare($sql);

// 执行插入语句
$stmt->execute(["test1","1234","13888888888"]);
$stmt->execute(["test2","1232","13888888888"]);
```

事务

事务（Transaction）是一组数据库操作的集合，这些操作要么全部成功执行，要么全部回滚（撤销）。事务提供了一种机制来确保数据库操作的一致性和完整性。

```php
try {
    // 开始事务
    $conn->beginTransaction();

    // 执行批量插入语句
    $sql = "INSERT INTO user (username, password, phone) VALUES (?,?,?)";
    $stmt = $conn->prepare($sql);

    $data = array(
        array("值1", "值2", "值3"),
        array("值4", "值5", "值6"),
        array("值7", "值8", "值9")
    );

    $insertedRows = 0;
    foreach ($data as $row) {
        $stmt->execute($row);
        $insertedRows += $stmt->rowCount();
    }

    // 提交事务
    $conn->commit();

    echo "批量插入成功";
} catch(PDOException $e) {
    // 回滚事务
    $conn->rollback();
    echo "插入失败: " . $e->getMessage();
}
```

作业：把PDO里没有用预处理的改为预处理方式。如果有时间，可以把上面的增删改查封装为一个DB类，方便调用。
后面我们实战的时候也会讲PDO类的二次封装和调用。

cookie session

cookie

cookie 常用于识别用户。
cookie 是一种服务器留在用户计算机上的小文件。
每当同一台计算机通过浏览器请求页面时，这台计算机将会发送 cookie。
通过 PHP，能够创建并取回 cookie 的值。

创建cookie

```
setcookie(name, value, expire, path, domain);
```

参数说明：

- name： Cookie的名称。
- value： Cookie的值。
- expire： Cookie的过期时间，可以是一个Unix时间戳（例如： time() + 3600 表示一小时后过期），或者是一个具体的日期时间字符串（例如： "2024-01-01 00:00:00" ）。
- path： 可选参数，指定Cookie的有效路径。默认为当前路径。
- domain： 可选参数，指定Cookie的有效域名。默认为空，表示当前域名。
- secure： 可选参数，指定是否仅通过安全的HTTPS连接传输Cookie。默认为 false 。
- httponly：可选参数，指定是否仅通过HTTP协议访问Cookie。默认为 false 。
-

```
$expire=time()+60*60*24*7;
setcookie("username", "maoshu", $expire);
setcookie("userid", "1", $expire);

setcookie("user[name]", "maoshu", $expire);
setcookie("user[phone]", "18611112222", $expire);
setcookie("user[age]", "35", $expire);
```

获取cookie

```
// 输出 cookie 值
if(isset($_COOKIE["username"])){
        echo $_COOKIE["username"];
}

// 查看所有 cookie
print_r($_COOKIE);
```

删除 Cookie

```
setcookie("username", "", time()-3600);
```

session

- 用于在服务器上存储关于用户会话（session）的信息，并且对于程序中的所有页面都是可用的。
- 会话信息是临时的，在用户离开网站后将被删除。
- 它允许在同一个用户的多个请求之间保持数据的状态，便于传递数据。
  Session 的工作机制是：为每个访客创建一个唯一的 id (UID)，并基于这个 UID 来存储变量。UID 存储在 cookie 中，或者通过 URL 进行传导。

启动session

通常，将该函数放在PHP文件的顶部

```
session_start();
```

存取 Session 变量

一旦Session启动，可以使用 `$_SESSION` 超全局变量来存储和访问Session数据。

```
// 存储 session 数据
$_SESSION['username']='maoshu';

//取值
echo $_SESSION['username'];
```

```
if(isset($_SESSION['views']))
{
    $_SESSION['views']=$_SESSION['views']+1;
}
else
{
    $_SESSION['views']=1;
}
echo "浏览量: ". $_SESSION['views'];
```

销毁 Session

```
if(isset($_SESSION['views']))
{
    unset($_SESSION['views']);
}
```

```
//销毁
session_destroy();
```

> 注意：session_destroy() 将重置 session，您将失去所有已存储的 session 数据

Session和Cookie的关系和区别

关系：

- Session和Cookie都是用于在不同请求之间保持数据的状态。
- Session使用Cookie来跟踪和标识用户，通过在Cookie中存储Session ID来关联服务器端的Session数据。

区别：

- 存储位置：Cookie数据存储在客户端的浏览器中，而Session数据存储在服务器端。
- 容量限制：Cookie的容量限制通常较小，一般为几KB，而Session的容量限制较大，通常取决于服务器的配置。
- 安全性：由于Cookie存储在客户端，可能会被篡改或窃取，因此存储敏感信息的安全性较低。相比之下，Session数据存储在服务器端，相对更安全。
- 生命周期：Cookie可以设置过期时间，可以在浏览器关闭后仍然保持，具有较长的生命周期。而Session通常在用户关闭浏览器或一段时间不活动后自动过期。
- 存储方式：Cookie以键值对的形式存储数据，可以在客户端进行读取和修改。Session数据存储在服务器端，客户端只保存了一个Session ID。

在实际应用中，Cookie通常用于存储一些较小、不敏感的数据，如用户首选项、购物车信息等。而Session通常用于存储用户的身份验证信息、敏感数据等，因为Session数据存储在服务器端，相对更安全。

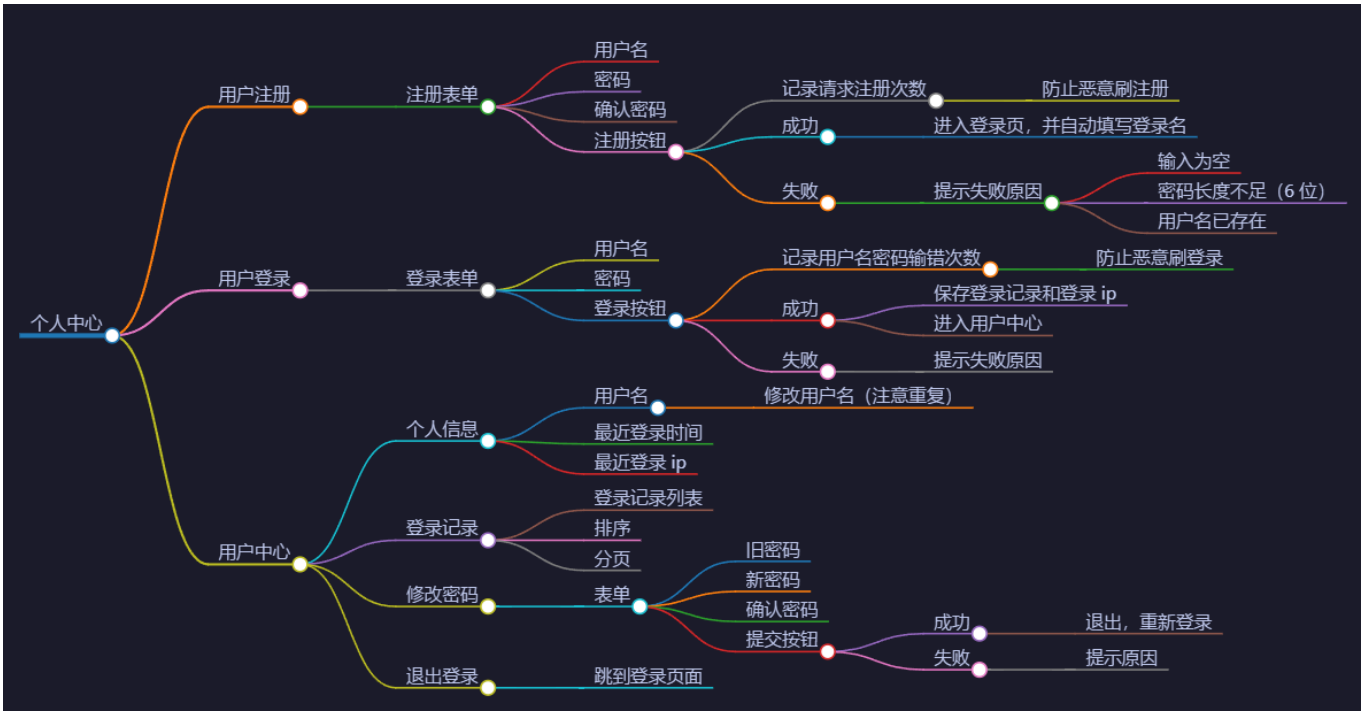
实战案例：个人中心

案例介绍：
我们会用最基础的原生 php 方式去做这个项目，不用任何前后端框架，考虑到一些学员没有前端基础，所以我会尽量少的使用 js，但会用到 fetch 请求等基础语句，并且 html 和 css 会有入门级的讲解（简单）；
我会尽可能多的使用学到的知识，以达到学以致用的效果；
我们会使用面向对象去对一些常用的方法进行封装，例如数据库操作，让大家学会在实际项目中使用类和对象。

讲解的内容：

- 什么是前端，什么是后端；什么是前台，什么是后台

- 页面搭建 (从 0 到 1)
- 什么是 json
- 简单的 Api 接口和请求
- Php 调试代码
- 封装类
- 文件包含
- 数据库的使用
- 用户状态



用户注册

用户登录

用户中心

个人信息

登录记录

修改密码

退出登录

实战补充知识

`str_repeat()` 和 `rtrim ()` `ltrim()` 函数

`floor()` 和 `ceil()` `round()` 函数

`md5()` 加密

die 和 exit 函数

常量 DIRECTORY_SEPARATOR

Pdo 补充知识

$stmt->closeCursor()If$(condition instanceof Closure)

`dirname ()`

`dirname` 函数用于返回指定路径的目录部分。它接受一个字符串参数，该参数是一个文件或目录的路径，并返回该路径的父级目录。

```php
$path = '/var/www/html/index.php';
$directory = dirname($path);
echo $directory;  // 输出: /var/www/html

$path = '/var/www/html/';
$directory = dirname($path);
echo $directory;  // 输出: /var/www

$path = '/var/www/html';
$directory = dirname($path);
echo $directory;  // 输出: /var/www
```

header ()函数

`header()` 函数用于发送原始的 HTTP 头信息。它通常用于在服务器响应中设置 HTTP 头，例如设置重定向、设置响应状态码、设置内容类型等。

```php
header(string $header, bool $replace = true, int $http_response_code = null): void
```

参数说明：

- $header： 要设置的 HTTP 头信息，以字符串形式表示。
- $replace （可选）： 指定是否替换之前设置的相同类型的 HTTP 头信息。默认为 true ， 表示替换。
- $http_response_code（可选）： 指定要设置的响应状态码。

```php
//重定向
header("Location: ./login.php");
//设置响应状态码:
header("HTTP/1.1 404 Not Found");
//设置内容类型
header("Content-Type: application/json");
```

注意： `header()` 函数必须在发送任何实际输出之前调用，包括 HTML 标记、空格、换行符等。如果在输出之后调用 `header()` 函数，将会导致 "headers already sent" 错误。

静态变量的使用

与类或函数相关联，而不是与对象或函数的实例相关联。
静态变量在整个脚本的执行过程中保持其值不变。

类

```php
class MyClass {
    public static $count = 0;
    public static function toObj()
    {
      return new self;
    }
    public function plusCount() {
        self::$count++;
        echo self::$count.PHP_EOL;
    }
}
MyClass::toObj()->plusCount();
MyClass::toObj()->plusCount();
MyClass::toObj()->plusCount();
echo MyClass::$count;
```

普通函数

> 静态变量只在函数的作用域内有效，不同函数中的同名静态变量是相互独立的

```php
function myFunction() {
    static $count = 0;

    $count++;
    echo $count.PHP_EOL;
}
```

```php
function mySecond() {
    static $count = 0;
    echo $count.PHP_EOL;
}
myFunction();
myFunction();
myFunction();

mySecond();
```

关于phpinfo()